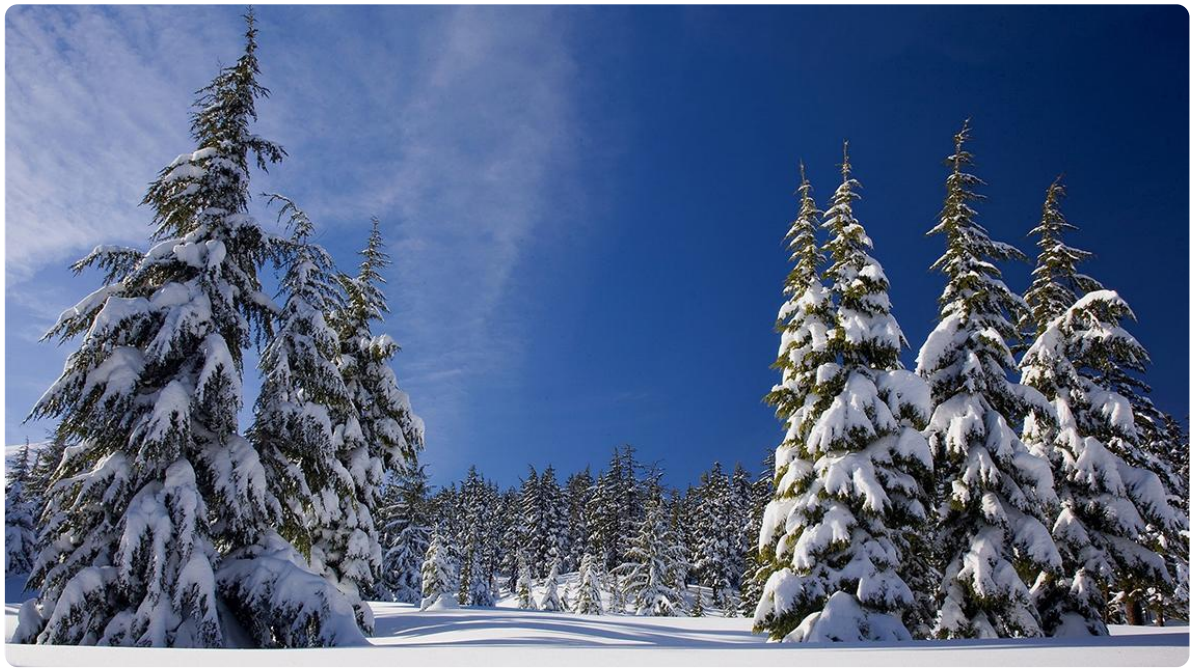

讲述：王庆友
时长 00:25:56 大小 24,331.58M

你好，我是欧创新。好久不见，今天我带着你期待的案例来了。

还记得我们在 [第 18 讲] 中用事件风暴完成的“在线请假考勤”项目的领域建模和微服务设计吗？今天我们就在这个项目的基础上看看，用DDD方法设计和开发出来的微服务代码到底是什么样的？点击 Github 获取完整代码，接下来的内容是我对代码的一个详解，期待能帮助你更好地实践我们这个专栏所学到的知识。

## 项目回顾

“在线请假考勤”项目中，请假的核心业务流程是：请假人填写请假单提交审批；根据请假人身份、请假类型和请假天数进行校验并确定审批规则；根据审批规则确定审批人，逐级提交上级审批，逐级核批通过则完成审批，否则审批不通过则退回申请人。

在 [第 18 讲] 的微服务设计中，我们已经拆分出了两个微服务：请假和考勤微服务。今天我们就围绕“请假微服务”来进行代码详解。微服务采用的开发语言和数据库分别是：Java、Spring boot 和 PostgreSQL。

## 请假微服务采用的DDD设计思想

请假微服务中用到了很多的DDD设计思想和方法，主要包括以下几个：

| 名称 | 具体内容 |
| --- | --- |
| 聚合的管理 | 聚合根、实体和值对象的关系 |
| 聚合数据的初始化和持久化 | 工厂和仓储模式 |
| 聚合的解耦 | 聚合代码的解耦、跨聚合的服务调用和对象解耦 |
| 领域事件管理 | 领域事件实体结构、持久化和事件发布 |
| DDD 分层架构 | 基础层、领域层、应用层和用户接口层的协作 |
| 服务的分层与协作 | 实体方法、领域服务、应用服务、接口服务，服务的组合和编排，跨多个聚合的服务管理和协同 |
| 对象的分层和转换 | DTO、DO 和 PO 等对象在不同层的转换和实现过程 |
| 微服务之间的访问 | 登录和认证服务 |

# 聚合中的对象

请假微服务包含请假（leave）、人员（person）和审批规则（rule）三个聚合。leave聚合完成请假申请和审核核心逻辑；person聚合管理人员信息和上下级关系；rule是一个单实体聚合，提供请假审批规则查询。

Leave是请假微服务的核心聚合，它有请假单聚合根leave、审批意见实体ApprovalInfo、请假申请人Applicant和审批人Approver值对象（它们的数据来源于person聚合），还有部分枚举类型，如请假类型LeaveType，请假单状态Status和审批状态类型ApprovalType等值对象。

下面我们通过代码来了解一下聚合根、实体以及值对象之间的关系。

## 1. 聚合根

聚合根leave中有属性、值对象、关联实体和自身的业务行为。Leave实体采用充血模型，有自己的业务行为，具体就是聚合根实体类的方法，如代码中的getDuration和addHistoryApprovalInfo等方法。

聚合根引用实体和值对象，它可以组合聚合内的多个实体，在聚合根实体类方法中完成复杂的业务行为，这种复杂的业务行为也可以在聚合领域服务里实现。但为了职责和边界清晰，我建议聚合要根据自身的业务行为在实体类方法中实现，而涉及多个实体组合才能实现的业务能力由领域服务完成。

下面是聚合根leave的实体类方法，它包含属性、对实体和值对象的引用以及自己的业务行为和方法。

```
public class Leave {

    @Autowired
    LeaveRepositoryInterface leaveRepositoryInterface;

    String id;
    Applicant applicant;
    Approver approver;
    LeaveType type;
    Status status;
    Date startTime;
    Date endTime;
    long duration;
    int leaderMaxLevel; //审批领导的最高级别
    ApprovalInfo currentApprovalInfo;
    List<ApprovalInfo> historyApprovalInfos;

    public long getDuration() {
        return endTime.getTime() - startTime.getTime();
    }

    public Leave addHistoryApprovalInfo(ApprovalInfo approvalInfo) {
        if (null == historyApprovalInfos)
            historyApprovalInfos = new ArrayList<>();
        this.historyApprovalInfos.add(approvalInfo);
        return this;
    }

    public Leave create(){
        this.setStatus(Status.APPROVING);
        this.setStartTime(new Date());
        return this;
}

//其它方法
}
```

### 2. 实体

审批意见实体ApprovalInfo被leave聚合根引用，用于记录审批意见，它有自己的属性和值对象，如approver等，业务逻辑相对简单。

```
public class ApprovalInfo {
    String approvalInfoId;
    Approver approver;
    ApprovalType approvalType;
    String msg;
    long time;
}
```

### 3. 值对象

在Leave聚合有比较多的值对象。

我们先来看一下审批人值对象Approver。这类值对象除了属性集之外，还可以有简单的数据查询和转换服务。Approver数据来源于person聚合，从person聚合获取审批人返回后，从person实体获取personID、personName和level等属性，重新组合为approver值对象，因此需要数据转换和重新赋

值。

Approver值对象同时被聚合根leave和实体approvalInfo引用。这类值对象的数据来源于其它聚合，不可修改，可重复使用。将这种对象设计为值对象而不是实体，可以提高系统性能，降低数据库实体关联的复杂度，所以我一般建议优先设计为值对象。

```
public class Approver {
    String personId;
    String personName;
    int level; //管理级别

    public static Approver fromPerson(Person person){
        Approver approver = new Approver();
        approver.setPersonId(person.getPersonId());
        approver.setPersonName(person.getPersonName());
        approver.setLevel(person.getRoleLevel());
        return approver;
    }
}
```

下面是枚举类型的值对象Status的代码。

```
public enum Status {
    APPROVING, APPROVED, REJECTED
}
```

这里你要记住一点，由于值对象只做整体替换、不可修改的特性，在值对象中基本不会有修改或新增的方法。

### 4. 领域服务

如果一个业务行为由多个实体对象参与完成，我们就将这部分业务逻辑放在领域服务中实现。领域服务与实体方法的区别是：实体方法完成单一实体自身的业务逻辑，是相对简单的原子业务逻辑，而领域服务则是多个实体组合出的相对复杂的业务逻辑。两者都在领域层，实现领域模型的核心业务能力。

一个聚合可以设计一个领域服务类，管理聚合内所有的领域服务。

请假聚合的领域服务类是LeaveDomainService。领域服务中会用到很多的DDD设计模式，比如：用工厂模式实现复杂聚合的实体数据初始化，用仓储模式实现领域层与基础层的依赖倒置和用领域事件实现数据的最终一致性等。

```
public class LeaveDomainService {

    @Autowired
    EventPublisher eventPublisher;
    @Autowired
    LeaveRepositoryInterface leaveRepositoryInterface;
    @Autowired
    LeaveFactory leaveFactory;

    @Transactional
    public void createLeave(Leave leave, int leaderMaxLevel, Approver approver) {
            leave.setLeaderMaxLevel(leaderMaxLevel);
```

```
            leave.setApprover(approver);
            leave.create();
    leaveRepositoryInterface.save(leaveFactory.createLeavePO(leave));
    LeaveEvent event = LeaveEvent.create(LeaveEventType.CREATE_EVENT, leave);
    leaveRepositoryInterface.saveEvent(leaveFactory.createLeaveEventPO(event));
    eventPublisher.publish(event);
    }

    @Transactional
    public void updateLeaveInfo(Leave leave) {
    LeavePO po = leaveRepositoryInterface.findById(leave.getId());
        if (null == po) {
                throw new RuntimeException("leave does not exist");
         }
     leaveRepositoryInterface.save(leaveFactory.createLeavePO(leave));
    }

    @Transactional
    public void submitApproval(Leave leave, Approver approver) {
       LeaveEvent event;
       if (ApprovalType.REJECT == leave.getCurrentApprovalInfo().getApprovalType()) {
       leave.reject(approver);
       event = LeaveEvent.create(LeaveEventType.REJECT_EVENT, leave);
       } else {
             if (approver != null) {
                 leave.agree(approver);
                 event = LeaveEvent.create(LeaveEventType.AGREE_EVENT, leave); } else {
                    leave.finish();
                    event = LeaveEvent.create(LeaveEventType.APPROVED_EVENT, leave);
                    }
           }
      leave.addHistoryApprovalInfo(leave.getCurrentApprovalInfo());
      leaveRepositoryInterface.save(leaveFactory.createLeavePO(leave));
      leaveRepositoryInterface.saveEvent(leaveFactory.createLeaveEventPO(event));
      eventPublisher.publish(event);
    }

    public Leave getLeaveInfo(String leaveId) {
    LeavePO leavePO = leaveRepositoryInterface.findById(leaveId);
    return leaveFactory.getLeave(leavePO);
    }

    public List<Leave> queryLeaveInfosByApplicant(String applicantId) {
        List<LeavePO> leavePOList = leaveRepositoryInterface.queryByApplicantId(applicantId);
    return leavePOList.stream().map(leavePO -> leaveFactory.getLeave(leavePO)).collect(Collect
    }

    public List<Leave> queryLeaveInfosByApprover(String approverId) {
    List<LeavePO> leavePOList = leaveRepositoryInterface.queryByApproverId(approverId);
    return leavePOList.stream().map(leavePO -> leaveFactory.getLeave(leavePO)).collect(Collect
    }
}
```

**领域服务开发时的注意事项：**

在领域服务或实体方法中，我们应尽量避免调用其它聚合的领域服务或引用其它聚合的实体或值对象，这种操作会增加聚合的耦合度。在微服务架构演进时，如果出现聚合拆分和重组，这种跨聚合的服务调用和对象引用，会变成跨微服务的操作，导致这种跨聚合的领域服务调用和对象引用失

效，在聚合分拆时会增加你代码解耦和重构的工作量。

以下是一段不建议使用的代码。在这段代码里Approver是leave聚合的值对象，它作为对象参数被传到person聚合的findNextApprover领域服务。如果在同一个微服务内，这种方式是没有问题的。但在架构演进时，如果person和leave两个聚合被分拆到不同的微服务中，那么leave中的Approver对象以及它的getPersonId()和fromPersonPO方法在person聚合中就会失效，这时你就需要进行代码重构了。

```
public class PersonDomainService {

   public Approver findNextApprover(Approver currentApprover, int leaderMaxLevel) {
   PersonPO leaderPO = personRepository.findLeaderByPersonId(currentApprover.getPersonId());
        if (leaderPO.getRoleLevel() > leaderMaxLevel) {
            return null;
        } else {
            return Approver.fromPersonPO(leaderPO);
        }
   }
}
```

那正确的方式是什么样的呢？在应用服务组合不同聚合的领域服务时，我们可以通过ID或者参数来传数，如单一参数currentApproverId。这样聚合之间就解耦了，下面是修改后的代码，它可以不依赖其它聚合的实体，独立完成业务逻辑。

```
public class PersonDomainService {

   public Person findNextApprover(String currentApproverId, int leaderMaxLevel) {
   PersonPO leaderPO = personRepository.findLeaderByPersonId(currentApproverId);
   if (leaderPO.getRoleLevel() > leaderMaxLevel) {
       return null;
    } else {
            return personFactory.createPerson(leaderPO);
      }
  }
}
```

# 领域事件

在创建请假单和请假审批过程中会产生领域事件。为了方便管理，我们将聚合内的领域事件相关的代码放在聚合的event目录中。领域事件实体在聚合仓储内完成持久化，但是事件实体的生命周期不受聚合根管理。

## 1. 领域事件基类DomainEvent

你可以建立统一的领域事件基类DomainEvent。基类包含：事件ID、时间戳、事件源以及事件相关的业务数据。

```
public class DomainEvent {
    String id;
    Date timestamp;
    String source;
    String data;
}
```

### 2. 领域事件实体

请假领域事件实体LeaveEvent继承基类DomainEvent。可根据需要扩展属性和方法，如leaveEventType。data字段中存储领域事件相关的业务数据，可以是XML或Json等格式。

```
public class LeaveEvent extends DomainEvent {
    LeaveEventType leaveEventType;
    public static LeaveEvent create(LeaveEventType eventType, Leave leave){
       LeaveEvent event = new LeaveEvent();
       event.setId(IdGenerator.nextId());
       event.setLeaveEventType(eventType);
       event.setTimestamp(new Date());
       event.setData(JSON.toJSONString(leave));
       return event;
    }
}
```

### 3. 领域事件的执行逻辑

一般来说，领域事件的执行逻辑如下：

第一步：执行业务逻辑，产生领域事件。

第二步：完成业务数据持久化。

```
leaveRepositoryInterface.save(leaveFactory.createLeavePO(leave));
```

第三步：完成事件数据持久化。

```
leaveRepositoryInterface.saveEvent(leaveFactory.createLeaveEventPO(event));
```

第四步：完成领域事件发布。

```
eventPublisher.publish(event);
```

以上领域事件处理逻辑代码详见LeaveDomainService中submitApproval领域服务，里面有请假提交审批事件的完整处理逻辑。

### 4. 领域事件数据持久化

为了保证事件发布方与事件订阅方数据的最终一致性和数据审计，有些业务场景需要建立数据对账机制。数据对账主要通过对源端和目的端的持久化数据比对，从而发现异常数据并进一步处理，保证数据最终一致性。

对于需要对账的事件数据，我们需设计领域事件对象的持久化对象PO，完成领域事件数据的持久化，如LeaveEvent事件实体的持久化对象LeaveEventPO。再通过聚合的仓储完成数据持久化：

```
leaveRepositoryInterface.saveEvent(leaveFactory.createLeaveEventPO(event))。
```

事件数据持久化对象LeaveEventPO格式如下：

```
public class LeaveEventPO {
    @Id
    @GenericGenerator(name = "idGenerator", strategy = "uuid")
    @GeneratedValue(generator = "idGenerator")
    int id;
    @Enumerated(EnumType.STRING)
    LeaveEventType leaveEventType;
    Date timestamp;
    String source;
    String data;
}
```

# 仓储模式

领域模型中DO实体的数据持久化是必不可少的，DDD采用仓储模式实现数据持久化，使得业务逻辑与基础资源逻辑解耦，实现依赖倒置。持久化时先完成DO与PO对象的转换，然后在仓储服务中完成PO对象的持久化。

## 1. DO与PO对象的转换

Leave聚合根的DO实体除了自身的属性外，还会根据领域模型引用多个值对象，如Applicant和Approver等，它们包含多个属性，如：personId、personName和personType等属性。

在持久化对象PO设计时，你可以将这些值对象属性嵌入PO属性中，或设计一个组合属性字段，以Json串的方式存储在PO中。

以下是leave的DO的属性定义：

```
public class Leave {
    String id;
    Applicant applicant;
    Approver approver;
    LeaveType type;
    Status status;
    Date startTime;
    Date endTime;
    long duration;
    int leaderMaxLevel;
    ApprovalInfo currentApprovalInfo;
    List<ApprovalInfo> historyApprovalInfos;
}

public class Applicant {
    String personId;
    String personName;
    String personType;
}

public class Approver {
    String personId;
    String personName;
    int level;
}
```

为了减少数据库表数量以及表与表的复杂关联关系，我们将leave实体和多个值对象放在一个

LeavePO中。如果以属性嵌入的方式，Applicant值对象在LeavePO中会展开为：applicantId、applicantName和applicantType三个属性。

以下为采用属性嵌入方式的持久化对象LeavePO的结构。

```
public class LeavePO {
    @Id
    @GenericGenerator(name="idGenerator", strategy="uuid")
    @GeneratedValue(generator="idGenerator")
    String id;
    String applicantId;
    String applicantName;
    @Enumerated(EnumType.STRING)
    PersonType applicantType;
    String approverId;
    String approverName;
    @Enumerated(EnumType.STRING)
    LeaveType leaveType;
    @Enumerated(EnumType.STRING)
    Status status;
    Date startTime;
    Date endTime;
    long duration;
    @Transient
    List<ApprovalInfoPO> historyApprovalInfoPOList;
}
```

## 2. 仓储模式

为了解耦业务逻辑和基础资源，我们可以在基础层和领域层之间增加一层仓储服务，实现依赖倒置。通过这一层可以实现业务逻辑和基础层资源的依赖分离。在变更基础层数据库的时候，你只要替换仓储实现就可以了，上层核心业务逻辑不会受基础资源变更的影响，从而实现依赖倒置。

一个聚合一个仓储，实现聚合数据的持久化。领域服务通过仓储接口来访问基础资源，由仓储实现完成数据持久化和初始化。仓储一般包含：仓储接口和仓储实现。

### 2.1仓储接口

仓储接口面向领域服务提供接口。

```
public interface LeaveRepositoryInterface {
   void save(LeavePO leavePO);
   void saveEvent(LeaveEventPO leaveEventPO);
   LeavePO findById(String id);
   List<LeavePO> queryByApplicantId(String applicantId);
   List<LeavePO> queryByApproverId(String approverId);
}
```

### 2.2仓储实现

仓储实现完成数据持久化和数据库查询。

```
@Repository
public class LeaveRepositoryImpl implements LeaveRepositoryInterface {

    @Autowired
    LeaveDao leaveDao;
    @Autowired
    ApprovalInfoDao approvalInfoDao;
    @Autowired
    LeaveEventDao leaveEventDao;

    public void save(LeavePO leavePO) {
        leaveDao.save(leavePO);
        approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
    }

    public void saveEvent(LeaveEventPO leaveEventPO){
        leaveEventDao.save(leaveEventPO);
    }

    @Override
    public LeavePO findById(String id) {
        return leaveDao.findById(id)
                .orElseThrow(() -> new RuntimeException("leave not found"));
    }

    @Override
    public List<LeavePO> queryByApplicantId(String applicantId) {
        List<LeavePO> leavePOList = leaveDao.queryByApplicantId(applicantId);
        leavePOList.stream()
                .forEach(leavePO -> {
                    List<ApprovalInfoPO> approvalInfoPOList = approvalInfoDao.queryByLeaveId(l
                    leavePO.setHistoryApprovalInfoPOList(approvalInfoPOList);
                });
        return leavePOList;
    }

    @Override
    public List<LeavePO> queryByApproverId(String approverId) {
        List<LeavePO> leavePOList = leaveDao.queryByApproverId(approverId);
        leavePOList.stream()
                .forEach(leavePO -> {
                    List<ApprovalInfoPO> approvalInfoPOList = approvalInfoDao.queryByLeaveId(l
                    leavePO.setHistoryApprovalInfoPOList(approvalInfoPOList);
                });
        return leavePOList;
    }
}
```

这里持久化组件采用了Jpa。

```
public interface LeaveDao extends JpaRepository<LeavePO, String> {
    List<LeavePO> queryByApplicantId(String applicantId);
    List<LeavePO> queryByApproverId(String approverId);
}
```

### 2.3仓储执行逻辑

以创建请假单为例，仓储的执行步骤如下。

第一步：仓储执行之前将聚合内DO会转换为PO，这种转换在工厂服务中完成：

```
leaveFactory.createLeavePO(leave)。
```

第二步：完成对象转换后，领域服务调用仓储接口：

```
leaveRepositoryInterface.save。
```

第三步：由仓储实现完成PO对象持久化。

代码执行步骤如下：

```
public void createLeave(Leave leave, int leaderMaxLevel, Approver approver) {
  leave.setLeaderMaxLevel(leaderMaxLevel);
  leave.setApprover(approver);
  leave.create();
  leaveRepositoryInterface.save(leaveFactory.createLeavePO(leave));
}
```

# 工厂模式

对于大型的复杂领域模型，聚合内的聚合根、实体和值对象之间的依赖关系比较复杂，这种过于复杂的依赖关系，不适合通过根实体构造器来创建。为了协调这种复杂的领域对象的创建和生命周期管理，在DDD里引入了工厂模式（Factory），在工厂里封装复杂的对象创建过程。

当聚合根被创建时，聚合内所有依赖的对象将会被同时创建。

工厂与仓储模式往往结对出现，应用于数据的初始化和持久化两类场景。

- DO对象的初始化：获取持久化对象PO，通过工厂一次构建出聚合根所有依赖的DO对象，完数据初始化。
- DO的对象持久化：将所有依赖的DO对象一次转换为PO对象，完成数据持久化。

下面代码是leave聚合的工厂类LeaveFactory。其中createLeavePO（leave）方法组织leave聚合的DO对象和值对象完成leavePO对象的构建。getLeave（leave）通过持久化对象PO构建聚合的DO对象和值对象，完成leave聚合DO实体的初始化。

```
public class LeaveFactory {

   public LeavePO createLeavePO(Leave leave) {
   LeavePO leavePO = new LeavePO();
   leavePO.setId(UUID.randomUUID().toString());
   leavePO.setApplicantId(leave.getApplicant().getPersonId());
   leavePO.setApplicantName(leave.getApplicant().getPersonName());
   leavePO.setApproverId(leave.getApprover().getPersonId());
   leavePO.setApproverName(leave.getApprover().getPersonName());
   leavePO.setStartTime(leave.getStartTime());
   leavePO.setStatus(leave.getStatus());
   List<ApprovalInfoPO> historyApprovalInfoPOList = approvalInfoPOListFromDO(leave);
   leavePO.setHistoryApprovalInfoPOList(historyApprovalInfoPOList);
   return leavePO;
}

   public Leave getLeave(LeavePO leavePO) {
   Leave leave = new Leave();
   Applicant applicant = Applicant.builder()
       .personId(leavePO.getApplicantId())
       .personName(leavePO.getApplicantName())
       .build();
   leave.setApplicant(applicant);
   Approver approver = Approver.builder()
       .personId(leavePO.getApproverId())
       .personName(leavePO.getApproverName())
       .build();
   leave.setApprover(approver);
   leave.setStartTime(leave.getStartTime());
   leave.setStatus(leave.getStatus());
   List<ApprovalInfo> approvalInfos = getApprovalInfos(leavePO.getHistoryApprovalInfoPOList())
   leave.setHistoryApprovalInfos(approvalInfos);
   return leave;
   }

//其它方法
}
```

## 服务的组合与编排

应用层的应用服务完成领域服务的组合与编排。一个聚合的应用服务可以建立一个应用服务类，管理聚合所有的应用服务。比如leave聚合有LeaveApplicationService，person聚合有PersonApplicationService。

在请假微服务中，有三个聚合：leave、person和rule。我们来看一下应用服务是如何跨聚合来进行服务的组合和编排的。以创建请假单createLeaveInfo应用服务为例，分为这样三个步骤。

第一步：根据请假单定义的人员类型、请假类型和请假时长从rule聚合中获取请假审批规则。这一步通过approvalRuleDomainService类的getLeaderMaxLevel领域服务来实现。

第二步：根据请假审批规则，从person聚合中获取请假审批人。这一步通过personDomainService类的findFirstApprover领域服务来实现。

第三步：根据请假数据和从rule和person聚合获取的数据，创建请假单。这一步通过

leaveDomainService类的createLeave领域服务来实现。

由于领域核心逻辑已经很好地沉淀到了领域层中，领域层的这些核心逻辑可以高度复用。应用服务只需要灵活地组合和编排这些不同聚合的领域服务，就可以很容易地适配前端业务的变化。因此应用层不会积累太多的业务逻辑代码，所以会变得很薄，代码维护起来也会容易得多。

以下是leave聚合的应用服务类。代码是不是非常得少?

```
public class LeaveApplicationService{

    @Autowired
    LeaveDomainService leaveDomainService;
    @Autowired
    PersonDomainService personDomainService;
    @Autowired
    ApprovalRuleDomainService approvalRuleDomainService;

    public void createLeaveInfo(Leave leave){
    //get approval leader max level by rule
    int leaderMaxLevel = approvalRuleDomainService.getLeaderMaxLevel(leave.getApplicant().getP
    //find next approver
    Person approver = personDomainService.findFirstApprover(leave.getApplicant().getPersonId()
    leaveDomainService.createLeave(leave, leaderMaxLevel, Approver.fromPerson(approver));
    }

    public void updateLeaveInfo(Leave leave){
    leaveDomainService.updateLeaveInfo(leave);
    }

    public void submitApproval(Leave leave){
    //find next approver
    Person approver = personDomainService.findNextApprover(leave.getApprover().getPersonId(),
    leaveDomainService.submitApproval(leave, Approver.fromPerson(approver));
    }

    public Leave getLeaveInfo(String leaveId){
        return leaveDomainService.getLeaveInfo(leaveId);
    }

    public List<Leave> queryLeaveInfosByApplicant(String applicantId){
    return leaveDomainService.queryLeaveInfosByApplicant(applicantId);
    }

    public List<Leave> queryLeaveInfosByApprover(String approverId){
    return leaveDomainService.queryLeaveInfosByApprover(approverId);
    }
}
```

**应用服务开发注意事项：**

为了聚合解耦和微服务架构演进，应用服务在对不同聚合领域服务进行编排时，应避免不同聚合的实体对象，在不同聚合的领域服务中引用，这是因为一旦聚合拆分和重组，这些跨聚合的对象将会失效。

在LeaveApplicationService中，leave实体和Applicant值对象分别作为参数被rule聚合和person聚合的领域服务引用，这样会增加聚合的耦合度。下面是不推荐使用的代码。

```
public class LeaveApplicationService{

  public void createLeaveInfo(Leave leave){
  //get approval leader max level by rule
  ApprovalRule rule = approvalRuleDomainService.getLeaveApprovalRule(leave);
  int leaderMaxLevel = approvalRuleDomainService.getLeaderMaxLevel(rule);
  leave.setLeaderMaxLevel(leaderMaxLevel);
  //find next approver
  Approver approver = personDomainService.findFirstApprover(leave.getApplicant(), leaderMaxLev
  leave.setApprover(approver);
  leaveDomainService.createLeave(leave);
  }
}
```

那如何实现聚合的解耦呢？我们可以将跨聚合调用时的对象传值调整为参数传值。一起来看一下调整后的代码，getLeaderMaxLevel由leave对象传值调整为personType，leaveType和duration参数传值。findFirstApprover中Applicant值对象调整为personId参数传值。

```
public class LeaveApplicationService{

  public void createLeaveInfo(Leave leave){
  //get approval leader max level by rule
  int leaderMaxLevel = approvalRuleDomainService.getLeaderMaxLevel(leave.getApplicant().getPer
  //find next approver
  Person approver = personDomainService.findFirstApprover(leave.getApplicant().getPersonId(),
  leaveDomainService.createLeave(leave, leaderMaxLevel, Approver.fromPerson(approver));
  }
}
```

在微服务演进和聚合重组时，就不需要进行聚合解耦和代码重构了。

# 微服务聚合拆分时的代码演进

如果请假微服务未来需要演进为人员和请假两个微服务，我们可以基于请假leave和人员person两个聚合来进行拆分。由于两个聚合已经完全解耦，领域逻辑非常稳定，在微服务聚合代码拆分时，聚合领域层的代码基本不需要调整。调整主要集中在微服务的应用服务中。

我们以应用服务createLeaveInfo为例，当一个微服务拆分为两个微服务时，看看代码需要做什么样的调整?

### 1. 微服务拆分前

createLeaveInfo应用服务的代码如下：

```
public void createLeaveInfo(Leave leave){

    //get approval leader max level by rule
    int leaderMaxLevel = approvalRuleDomainService.getLeaderMaxLevel(leave.getApplicant().getP
    //find next approver
    Person approver = personDomainService.findFirstApprover(leave.getApplicant().getPersonId()
    leaveDomainService.createLeave(leave, leaderMaxLevel, Approver.fromPerson(approver));
}
```

## 2. 微服务拆分后

leave和person两个聚合随微服务拆分后，createLeaveInfo应用服务中下面的代码将会变成跨微服务调用。

```
Person approver = personDomainService.findFirstApprover(leave.getApplicant().getPersonId(), le
```

由于跨微服务的调用是在应用层完成的，我们只需要调整createLeaveInfo应用服务代码，将原来微服务内的服务调用personDomainService.findFirstApprover修改为跨微服务的服务调用：personFeignService. findFirstApprover。

同时新增ApproverAssembler组装器和PersonResponse的DTO对象，以便将person微服务返回的person DTO对象转换为approver值对象。

```
// PersonResponse为调用微服务返回结果的封装
//通过personFeignService调用Person微服务用户接口层的findFirstApprover facade接口
PersonResponse approverResponse = personFeignService. findFirstApprover(leave.getApplicant().g
Approver approver = ApproverAssembler.toDO(approverResponse);
```

在原来的person聚合中，由于findFirstApprover领域服务已经逐层封装为用户接口层的Facade接口，所以person微服务不需要做任何代码调整，只需将PersonApi的findFirstApprover Facade服务，发布到API网关即可。

如果拆分前person聚合的findFirstApprover领域服务，没有被封装为Facade接口，我们只需要在person微服务中按照以下步骤调整即可。

第一步：将person聚合PersonDomainService类中的领域服务findFirstApprover封装为应用服务findFirstApprover。

```
@Service
public class PersonApplicationService {

  @Autowired
  PersonDomainService personDomainService;

  public Person findFirstApprover(String applicantId, int leaderMaxLevel) {
  return personDomainService.findFirstApprover(applicantId, leaderMaxLevel);
  }
}
```

第二步：将应用服务封装为Facade服务，并发布到API网关。

```
@RestController
@RequestMapping("/person")
@Slf4j
public class PersonApi {

  @Autowired
  @GetMapping("/findFirstApprover")
  public Response findFirstApprover(@RequestParam String applicantId, @RequestParam int leader
  Person person = personApplicationService.findFirstApprover(applicantId, leaderMaxLevel);
          return Response.ok(PersonAssembler.toDTO(person));
  }
}
```

# 服务接口的提供

用户接口层是前端应用与微服务应用层的桥梁，通过Facade接口封装应用服务，适配前端并提供灵活的服务，完成DO和DTO相互转换。

当应用服务接收到前端请求数据时，组装器会将DTO转换为DO。当应用服务向前端返回数据时，组装器会将DO转换为DTO。

## 1. facade接口

facade接口可以是一个门面接口实现类，也可以是门面接口加一个门面接口实现类。项目可以根据前端的复杂度进行选择，由于请假微服务前端功能相对简单，我们就直接用一个门面接口实现类来实现就可以了。

```
public class LeaveApi {
  @PostMapping
  public Response createLeaveInfo(LeaveDTO leaveDTO){
          Leave leave = LeaveAssembler.toDO(leaveDTO);
          leaveApplicationService.createLeaveInfo(leave);
          return Response.ok();
  }

  @PostMapping("/query/applicant/{applicantId}")
  public Response queryByApplicant(@PathVariable String applicantId){
  List<Leave> leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
  List<LeaveDTO> leaveDTOList = leaveList.stream().map(leave -> LeaveAssembler.toDTO(leave)).c
          return Response.ok(leaveDTOList);
  }

//其它方法
}
```

## 2. DTO数据组装

组装类（Assembler）： 负责将应用服务返回的多个DO对象组装为前端DTO对象，或将前端请求的DTO对象转换为多个DO对象，供应用服务作为参数使用。组装类中不应有业务逻辑，主要负责格式转换、字段映射等。Assembler往往与DTO同时存在。LeaveAssembler完成请假DO和DTO数据相互转换。

```
public class LeaveAssembler {

    public static LeaveDTO toDTO(Leave leave){
        LeaveDTO dto = new LeaveDTO();
        dto.setLeaveId(leave.getId());
        dto.setLeaveType(leave.getType().toString());
        dto.setStatus(leave.getStatus().toString());
        dto.setStartTime(DateUtil.formatDateTime(leave.getStartTime()));
        dto.setEndTime(DateUtil.formatDateTime(leave.getEndTime()));
        dto.setCurrentApprovalInfoDTO(ApprovalInfoAssembler.toDTO(leave.getCurrentApprovalInfo
        List<ApprovalInfoDTO> historyApprovalInfoDTOList = leave.getHistoryApprovalInfos()
                .stream()
                .map(historyApprovalInfo -> ApprovalInfoAssembler.toDTO(leave.getCurrentApprov
                .collect(Collectors.toList());
        dto.setHistoryApprovalInfoDTOList(historyApprovalInfoDTOList);
        dto.setDuration(leave.getDuration());
        return dto;
    }

    public static Leave toDO(LeaveDTO dto){
        Leave leave = new Leave();
        leave.setId(dto.getLeaveId());
        leave.setApplicant(ApplicantAssembler.toDO(dto.getApplicantDTO()));
        leave.setApprover(ApproverAssembler.toDO(dto.getApproverDTO()));
        leave.setCurrentApprovalInfo(ApprovalInfoAssembler.toDO(dto.getCurrentApprovalInfoDTO(
        List<ApprovalInfo> historyApprovalInfoDTOList = dto.getHistoryApprovalInfoDTOList()
                .stream()
                .map(historyApprovalInfoDTO -> ApprovalInfoAssembler.toDO(historyApprovalInfoD
                .collect(Collectors.toList());
        leave.setHistoryApprovalInfos(historyApprovalInfoDTOList);
        return leave;
    }
}
```

DTO类：包括requestDTO和responseDTO两部分。

DTO应尽量根据前端展示数据的需求来定义，避免过多地暴露后端业务逻辑。尤其对于多渠道场景，可以根据渠道属性和要求，为每个渠道前端应用定义个性化的DTO。由于请假微服务相对简单，我们可以用leaveDTO代码做个示例。

```
@Data
public class LeaveDTO {
    String leaveId;
    ApplicantDTO applicantDTO;
    ApproverDTO approverDTO;
    String leaveType;
    ApprovalInfoDTO currentApprovalInfoDTO;
    List<ApprovalInfoDTO> historyApprovalInfoDTOList;
    String startTime;
    String endTime;
    long duration;
    String status;
}
```

## 总结

今天我们了解了用DDD开发出来的微服务代码到底是什么样的。你可以将这些核心设计思想逐步引入到项目中去，慢慢充实自己的DDD知识体系。我还想再重点强调的是：由于架构的演进，微服务与生俱来就需要考虑聚合的未来重组。因此微服务的设计和开发要做到未雨绸缪，而这最关键的就是解耦了。

**聚合与聚合的解耦：**当多个聚合在同一个微服务时，很多传统架构开发人员会下意识地引用其他聚合的实体和值对象，或者调用其它聚合的领域服务。因为这些聚合的代码在同一个微服务内，运行时不会有问题，开发效率似乎也更高，但这样会不自觉地增加聚合之间的耦合。在微服务架构演进时，如果聚合被分别拆分到不同的微服务中，原来微服务内的关系就会变成跨微服务的关系，原来微服务内的对象引用或服务调用将会失效。最终你还是免不了要花大量的精力去做聚合解耦。虽然前期领域建模和边界划分得很好，但可能会因为开发稍不注意，而导致解耦工作前功尽弃。

**微服务内各层的解耦：**微服务内有四层，在应用层和领域层组成核心业务领域的两端，有两个缓冲区或数据转换区。前端与应用层通过组装器实现DTO和DO的转换，这种适配方式可以更容易地响应前端需求的变化，隐藏核心业务逻辑的实现，保证核心业务逻辑的稳定，实现核心业务逻辑与前端应用的解耦。而领域层与基础层通过仓储和工厂模式实现DO和PO的转换，实现应用逻辑与基础资源逻辑的解耦。

最后我想说，DDD知识体系虽大，但你可以根据企业的项目场景和成本要求，逐步引入适合自己的DDD方法和技术，建立适合自己的DDD开发模式和方法体系。

这一期的加餐到这就结束了，希望你能对照完整代码认真阅读今天的内容，有什么疑问，欢迎在留言区与我交流！



精选留言(44)


Miss

偶是编辑，这篇加餐比较长～作者会抽周末的时间把音频讲解为大家补上。感谢等待！

2020-1-2

CN....

老师好，浏览代码有两点疑惑
1，我们通常会认为加了事务注解就尽量避免除数据库外的其他调用，但是代码中在领域服务中的方法中发送mq，而且是在有事务注解的方法中，这里是基于什么考虑
2，消费mq的逻辑应该属于那一层
谢谢

> 作者回复：1、这个主要是考虑业务数据，事件数据持久化和发送消息队列同时能够成功，避免出现数据不一致的情况。当然也可以只在业务数据和事件数据持久化增加事务，如果消息队列发送不成功，还可以从事件表中获取数据再次发送。
>
> 2、消息订阅方一般在应用层监听和接受事件数据。

2020-1-6

阿玛铭

欧老师的回马枪猝不及防

> 作者回复：🅰🅰，舍不得跟大家告别。

2020-1-2

盲僧

太棒了，这个案例太精彩

> 作者回复：🅰🅰

2020-1-3

沉迷学习Zzz

老师，我看了下你的示例，不是说领域层和持久化数据库之间加了一个仓储层吗，为什么仓储的实现还是放在领域层呢，不应该放在基础设施层吗

> 作者回复：我在目录结构那一章讲了。仓储实现理论上是基础层的内容。放领域层的主要目的是考虑到以后微服务的演进代码拆包和打包的便利性。因为一个聚合一个仓储，如果聚合需要进行重新拆分和组合，你就可以将聚合目录内所有代码作为一个包打包重新组合。

2020-3-6

zj

有个问题我还是想问问欧老师，在工作中我遇到这样的一个场景，KA商品推广实体，很多场景都可以用领域服务实现，但是有个场景是查询KA商品推广列表，列表上还要有每个KA商品的推广效果数据（点击量，曝光量，交易额，商详页UV），这个推广效果数据放在领域服务中去查询总感觉不合适，老师有什么好的建议

作者回复：基于聚合根的查询不太适合进行复杂的查询。

对于复杂的查询你可以通过应用服务，在仓储实现中直接用SQL查询也是可以的。

2020-1-7

切糕

老师，有个问题请教下：
应用层负责领域服务的组织和编排，我看示例代码的事务控制在领域服务层，多个领域服务如何确保事务的一致性。
示例：

```
public void createLeaveInfo(Leave leave){
//get approval leader max level by rule
int leaderMaxLevel = approvalRuleDomainService.getLeaderMaxLevel(leave.getApplicant().getPersonType(), leave.getType().toString(), leave.getDuration());
//find next approver
Person approver = personDomainService.findFirstApprover(leave.getApplicant().getPersonId(), leaderMaxLevel);
leaveDomainService.createLeave(leave, leaderMaxLevel, Approver.fromPerson(approver));
}
```

作者回复：这个领域服务的事务控制主要是保证写业务表和事件表以及发布到消息队列，这三步能够保持数据一致。聚合之间或微服务之间可以在应用服务采用事务机制。

2020-1-6

陈四丰

感谢欧老师的加餐，受益匪浅。

实际操作中，我产生了一个疑问。是不是每个微服务都要有自己单独的数据库？比如本节案例：“请假”和“考勤”是两个微服务，那么就应该有两个独立的数据库去对应。(database: leave和checking)?

多谢指教!!!

作者回复：如果按照领域建模的方式来设计微服务的话，我们可以很容易的对对象进行聚合以及实体归类，所以也很容易的实现数据之间的解耦，没有特殊考虑的话，建议一个微服务一个数据库。

2020-1-5

zj

这篇文章刚下班，还没来得及看，等会下班到家再一饱眼福。

2020-1-2

miniluo

最近公司使用sofa来重构已有的业务代码，对比了老师这个，虽然大体上都差不多，但是你这个各层的划分还比较清晰。得到不少的领悟，感谢！

> 作者回复：谢谢，欢迎多交流。

2020-3-17

Geek_88604f

老师，同一个聚会内的两个实体之间也不能互相引用，只能通过领域服务来协调？这样的话，一个聚合内除了聚合根外，其他的实体都只能引用一些基本的对象，string、timestamp等？

> 作者回复：实体之间可以引用的，这里的协同主要强调实体之间的业务逻辑，涉及到多实体的业务逻辑组合的事情。

2020-3-8

克制的民工

这一讲精彩！要好好消化一下。
再请教一个问题：前后端如果采用restful接口，请问restful api和facade 接口是什么关系？在facade 接口的基础上再封装restful api吗？在DDM分层架构中，restful api属于那一层呢。谢谢

> 作者回复：Facade接口就是restful api，facade接口发布到API网关，供前端调用。

2020-3-6

marker

事件存储和业务存储可以用aop封装一下,总体非常好

2020-3-2

修冶

老师你好，Dao相关的操作能放到实体对象里吗？

作者回复：数据库相关的操作不要放在领域层，放在仓储的实现里面比较合适。这样才能解耦领域逻辑和数据库逻辑。

2020-2-25

加载中……

老师好，针对github上的代码有个疑问想请教下：
关于持久化的操作是不是也应该在DO里面(比如Person类) Person.findById()里面再调用repository。我看有些DDD实践是这样描述的，不过这样实现好像会复杂些。
我看咱们的github上的代码DO里面没有关于CRUD对应的操作语义，这些都放到DomainService来操作了。
问题：这些持久化相关的语义操作 用不用放到DO里面实现呢?

作者回复：持久化的操作应该是PO。通过仓储服务来实现。CRUD用JPA的话，不需要你单独写方法的。持久化的操作都是放在仓储实现里的，不应该放DO里面。

2020-2-18

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

作者回复：有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

作者回复：比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以了。

同一个聚合内持久化并发建议通过数据库SQL

来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗？

作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。

do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ？

作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

Geek_88604f

老师，同一个聚会内的两个实体之间也不能互相引用，只能通过领域服务来协调？这样的话，一个聚合内除了聚合根外，其他的实体都只能引用一些基本的对象，string、timestamp等？

作者回复：实体之间可以引用的，这里的协同主要强调实体之间的业务逻辑，涉及到多实体的业务逻辑组合的事情。

2020-3-8

克制的民工

这一讲精彩！要好好消化一下。
再请教一个问题：前后端如果采用restful接口，请问restful api和facade 接口是什么关系？在facade 接口的基础上再封装restful api吗？在DDM分层架构中，restful api属于那一层呢。谢谢

> 作者回复：Facade接口就是restful api，facade接口发布到API网关，供前端调用。

2020-3-6

marker

事件存储和业务存储可以用aop封装一下,总体非常好

2020-3-2

修冶

老师你好，Dao相关的操作能放到实体对象里吗?

> 作者回复：数据库相关的操作不要放在领域层，放在仓储的实现里面比较合适。这样才能解耦领域逻辑和数据库逻辑。

2020-2-25

加载中……

老师好，针对github上的代码有个疑问想请教下：
关于持久化的操作是不是也应该在DO里面(比如Person类) Person.findById()里面再调用repository。我看有些DDD实践是这样描述的，不过这样实现好像会复杂些。
我看咱们的github上的代码DO里面没有关于CRUD对应的操作语义，这些都放到DomainService来操作了。
问题：这些持久化相关的语义操作 用不用放到DO里面实现呢?

> 作者回复：持久化的操作应该是PO。通过仓储服务来实现。CRUD用JPA的话，不需要你单独写方法的。
> 持久化的操作都是放在仓储实现里的，不应该放DO里面。

2020-2-18

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

作者回复：有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

作者回复：比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以了。
同一个聚合内持久化并发建议通过数据库SQL
来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗?

作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。
do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,

对于每一层, 也是需要面向接口编程的吧 ？

> 作者回复 : 是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

> 作者回复 : 微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

> 作者回复 : 麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

> 作者回复 : 一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助

guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13



uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，👍👍👍👍👍👍👍👍👍👍👍👍👍👍

作者回复：过来人👍👍。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

克制的民工

这一讲精彩！要好好消化一下。
再请教一个问题：前后端如果采用restful接口，请问restful api和facade 接口是什么关系？在facade 接口的基础上再封装restful api吗？在DDM分层架构中，restful api属于那一层呢。谢谢

作者回复：Facade接口就是restful api，facade接口发布到API网关，供前端调用。

2020-3-6

marker

事件存储和业务存储可以用aop封装一下,总体非常好

2020-3-2

修冶

老师你好，Dao相关的操作能放到实体对象里吗?

> 作者回复：数据库相关的操作不要放在领域层，放在仓储的实现里面比较合适。这样才能解耦领域逻辑和数据库逻辑。

2020-2-25

加载中……

老师好，针对github上的代码有个疑问想请教下：
关于持久化的操作是不是也应该在DO里面(比如Person类) Person.findById()里面再调用repository
。我看有些DDD实践是这样描述的，不过这样实现好像会复杂些。
我看咱们的github上的代码DO里面没有关于CRUD对应的操作语义，这些都放到DomainService来操作了。
问题：这些持久化相关的语义操作 用不用放到DO里面实现呢?

> 作者回复：持久化的操作应该是PO。通过仓储服务来实现。CRUD用JPA的话，不需要你单独写方法的。持久化的操作都是放在仓储实现里的，不应该放DO里面。

2020-2-18

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

> 作者回复：有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。

但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

> 作者回复：比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以了。
> 同一个聚合内持久化并发建议通过数据库SQL
> 来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗?

> 作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。
> do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ？

> 作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账

单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

> 作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复：JPA可以自动创建的。

2020-1-13



uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️

作者回复：过来人🀄️🀄️。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

marker

事件存储和业务存储可以用aop封装一下,总体非常好

2020-3-2

修冶

老师你好，Dao相关的操作能放到实体对象里吗?

作者回复：数据库相关的操作不要放在领域层，放在仓储的实现里面比较合适。这样才能解耦领域逻辑和数据库逻辑。

2020-2-25

加载中……

老师好，针对github上的代码有个疑问想请教下：
关于持久化的操作是不是也应该在DO里面(比如Person类) Person.findById()里面再调用repository
。我看有些DDD实践是这样描述的，不过这样实现好像会复杂些。
我看咱们的github上的代码DO里面没有关于CRUD对应的操作语义，这些都放到DomainService来操作了。
问题：这些持久化相关的语义操作 用不用放到DO里面实现呢?

作者回复：持久化的操作应该是PO。通过仓储服务来实现。CRUD用JPA的话，不需要你单独写方法的。持久化的操作都是放在仓储实现里的，不应该放DO里面。

2020-2-18

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

作者回复：有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

作者回复：比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以了。

同一个聚合内持久化并发建议通过数据库SQL

来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗？

作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。

do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ？

作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

> 作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

> 作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

> 作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

> 作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复：JPA可以自动创建的。

2020-1-13



uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

> 作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，👍👍👍👍👍👍👍👍👍👍👍👍👍👍

作者回复：过来人👍👍。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

修冶

老师你好，Dao相关的操作能放到实体对象里吗?

作者回复：数据库相关的操作不要放在领域层，放在仓储的实现里面比较合适。这样才能解耦领域逻辑和数据库逻辑。

2020-2-25

加载中……

老师好，针对github上的代码有个疑问想请教下：
关于持久化的操作是不是也应该在DO里面(比如Person类) Person.findById()里面再调用repository
。我看有些DDD实践是这样描述的，不过这样实现好像会复杂些。
我看咱们的github上的代码DO里面没有关于CRUD对应的操作语义，这些都放到DomainService来操作了。
问题：这些持久化相关的语义操作 用不用放到DO里面实现呢?

> 作者回复：持久化的操作应该是PO。通过仓储服务来实现。CRUD用JPA的话，不需要你单独写方法的。
>
> 持久化的操作都是放在仓储实现里的，不应该放DO里面。

2020-2-18

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

> 作者回复：有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太
>
> 大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

> 作者回复：比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以
>
> 了。
>
> 同一个聚合内持久化并发建议通过数据库SQL
>
> 来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗？

作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。

do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ？

作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚

合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13

uangguan

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️

作者回复：过来人🅰️🅰️。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

加载中……

老师好，针对github上的代码有个疑问想请教下：

关于持久化的操作是不是也应该在DO里面(比如Person类) Person.findById()里面再调用repository
。我看有些DDD实践是这样描述的，不过这样实现好像会复杂些。
我看咱们的github上的代码DO里面没有关于CRUD对应的操作语义，这些都放到DomainService来操作了。
问题：这些持久化相关的语义操作 用不用放到DO里面实现呢?

> 作者回复 : 持久化的操作应该是PO。通过仓储服务来实现。CRUD用JPA的话，不需要你单独写方法的。
>
> 持久化的操作都是放在仓储实现里的，不应该放DO里面。

2020-2-18

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

> 作者回复 : 有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太
>
> 大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

> 作者回复 : 比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以
>
> 了。
>
> 同一个聚合内持久化并发建议通过数据库SQL
>
> 来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗?

> 作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。
>
> do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ？

> 作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

> 作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

> 作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复 : JPA可以自动创建的。

2020-1-13

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复 : 你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

> 作者回复 : 复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂

> 作者回复 : 过来人🏂🏂。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

public void save(LeavePO leavePO) {

```
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

mliu

老师你好，上面说一个聚合内的业务逻辑都在一个 领域服务类中实现(xxxxDomanService),那如果业务复杂的话，会不会导致这个类比较大，比较臃肿?

> 作者回复 : 有可能的，所以设计的时候要设计小聚合。太大的话，可能会变成小单体。如果领域服务太大，也可以将比较大的领域服务独立为一个领域服务类。

2020-2-16

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

> 作者回复 : 比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以了。
> 同一个聚合内持久化并发建议通过数据库SQL
> 来实现。

2020-2-16

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗？

> 作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。
>
> do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ？

> 作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

> 作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

> 作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复 : 一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复 : 是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复 : 我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复：JPA可以自动创建的。

2020-1-13

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

> 作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂

> 作者回复：过来人🏂。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

> 作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

> 作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

> 作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

> 作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

> 作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么？

> 作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

蒙面超人

感谢老师的分享，在实践过程中遇到些疑问，望解答：
1. 类似Leave这样的聚合根（真实业务场景可能会更大一些，大小的控制也比较考验水平），会初始化查询好几张表，性能会下降，这个量不大可以接受。
但是 List leaveList = leaveApplicationService.queryLeaveInfosByApplicant(applicantId);
这类查询请假集合，针对展示层，经由领域层，聚合控制不好，性能会显著下降。是不是直接由仓储输出DTO更好呢?
2. submitApproval 可能产生的并发变更问题，逻辑过程经由
a.加载leave聚合根对象
b.判断ApprovalType.REJECT == leave.getCurrentApprovalInfo()状态
c.逻辑处理以及持久化
leave由于版本比较旧，造成b的判断问题，而产生并发问题。
这种如何处理比较好?
是sql 额外 控制状态转变的逻辑，还是分布式锁在应用层控制?

> 作者回复：比较复杂的查询逻辑，建议你不过领域层，直接从应用服务按照传统的查询方式设计就可以了。
> 同一个聚合内持久化并发建议通过数据库SQL
> 来实现。

2020-2-16

allen

alien

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗？

> 作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。
>
> do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ?

> 作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

> 作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

> 作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复：JPA可以自动创建的。

2020-1-13

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

> 作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂

> 作者回复：过来人🏂🏂。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

public void save(LeavePO leavePO) {

```
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么?

> 作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

> 作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

allen

为什么在仓储实现层加注解@Repository呢？这个注解不都是应用在dao上吗？还有就是仓储实现都是对po的操作，do与po的转换是在领域服务层完成吗?

> 作者回复：@repository注解在传统三层结构一般是放在dao那层表示持久化层，属于习惯，没有规范要求，仓储需要注入肯定需要注解，用@Component也可以，习惯上用@Repository表示持久化相关的更合适。
>
> do和po的转换放仓储实现里面也是可以的，放这里更接近数据。DO和PO转换因为涉及到比较多的实体操作，所以放到工厂里去了。

2020-2-12

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧 ?

作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13



uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️🀄️

作者回复：过来人🀄️🀄️。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么?

作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰🅰

作者回复：🅰🅰，久等了。

2020-1-3

嘉嘉

老师, 请问下,
对于每一层, 也是需要面向接口编程的吧？

作者回复：是的，逐层封装。

2020-2-10

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

> 作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复：JPA可以自动创建的。

2020-1-13



uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

> 作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，👍👍👍👍👍👍👍👍👍👍👍👍👍👍

作者回复：过来人👍👍。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么？

作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰️🅰️

作者回复：🅰️🅰️，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

作者回复：是的，DDD不是一个人的事情。

2020-1-2

微光

棒，值得学习->实践->再学习

2020-2-5

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊?

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

> 作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋

> 作者回复：过来人🙋🙋。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

> 作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么？

> 作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

> 作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰🅰

> 作者回复：🅰🅰，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

> 作者回复：是的，DDD不是一个人的事情。

2020-1-2

观弈道人

终于等到了，谢谢大佬。

2020-1-2

微光

有个疑问，如果是微服务之间调用，为了减少重复编码和统一性，是不是应该把DTO和FeginClient打成jar包发布到仓库，让调用方引用啊？

作者回复：微服务之间主要通过微服务网关来调用，Facade接口直接发布就可以了。

2020-2-5

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：

1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13



uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，👍👍👍👍👍👍👍👍👍👍👍👍👍👍

作者回复：过来人👍👍。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么?

作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！

作者回复：🅰🅰，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

作者回复：是的，DDD不是一个人的事情。

2020-1-2

观弈道人

终于等到了，谢谢大佬。

2020-1-2

和光同尘

请问数据对账机制，是如何知道有数据缺损，不一致的？再者，发现不一致的数据如何纠正。3Q

作者回复：可以基于源端持久化和目的端持久化数据来比对，发现异常数据。处理方式有很多，发现问题后可以重传或者人工处理。

2020-1-2

静静聆听

老师，Leave是个实体类，怎么能用@Autowired注入Bean的

作者回复：麻烦告诉一下是哪个类，大概哪个位置。谢谢

2020-1-28

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会

修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋

作者回复：过来人🙋。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
```

```
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么？

> 作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

> 作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰🅰

> 作者回复：🅰🅰，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

> 作者回复：是的，DDD不是一个人的事情。

2020-1-2

观弈道人

终于等到了，谢谢大佬。

2020-1-2

和光同尘

请问数据对账机制，是如何知道有数据缺损，不一致的？再者，发现不一致的数据如何纠正。3Q

作者回复：可以基于源端持久化和目的端持久化数据来比对，发现异常数据。处理方式有很多，发现问题后可以重传或者人工处理。

2020-1-2

Jxin

1.感谢栏主的demo。劳烦了。

2.毕竟是用于大概了解ddd的demo，各中实现比较简单，并没有很讲究设计原则。如果有下一个专栏再探讨。

3.认真看完栏主代码实现，跟自己的实现规范差别还是挺多的。各有利弊就不多赘述了。

4.期待下次相见。

作者回复：DDD的主要目标是为了边界划分和解耦，而实现这个目标有很多的手段。目标只有一个，而手段会有很多个，只要能达到这个目标，可以选择适合自己的手段，咱们的主要区别可能在手段上有差异。有机会向你学习哈。

2020-1-2

若水

老师您好，这样 设计一个组合属性字段，以 Json 串的方式存储在 PO，会存在如下两个问题。①、数据修改，对应记录值不会修改。 ②、这样设计一个大字段存储，引发数据库查询的效率问题。 这两个问题您是怎么看待呢

作者回复：一般对于有查询和统计要求的值对象，不建议做成json串存储。其实如果这部分数据在处理后，加载到数据平台，也不影响查询使用。这类值对象的修改需要对应的前端操作，所以其它地方修改并不会修改值对象的值，这也是一种正常的需求，它记录业务发生那一刻的数据快照，不会因为外部数据修改而导致无法记录业务产生那一刻的真实数据。

2020-1-22

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13

uangguan

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂🏂

作者回复：过来人🏂。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么?

作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰🅰

作者回复：🅰🅰，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

作者回复：是的，DDD不是一个人的事情。

2020-1-2

观弈道人

终于等到了，谢谢大佬。

2020-1-2

和光同尘

请问数据对账机制，是如何知道有数据缺损，不一致的？再者，发现不一致的数据如何纠正。3Q

作者回复：可以基于源端持久化和目的端持久化数据来比对，发现异常数据。处理方式有很多，发现问题后可以重传或者人工处理。

2020-1-2

Jxin

1.感谢栏主的demo。劳烦了。

2.毕竟是用于大概了解ddd的demo，各中实现比较简单，并没有很讲究设计原则。如果有下一个专栏再探讨。

3.认真看完栏主代码实现，跟自己的实现规范差别还是挺多的。各有利弊就不多赘述了。

4.期待下次相见。

> 作者回复：DDD的主要目标是为了边界划分和解耦，而实现这个目标有很多的手段。目标只有一个，而手段会有很多个，只要能达到这个目标，可以选择适合自己的手段，咱们的主要区别可能在手段上有差异。有机会向你学习哈。

2020-1-2

深山小书童

终于等到了，欧老师威武

2020-1-2

focus

如果是内部事件，与外部服务无关，还需要定义listener，publish发布的是内部事件；这样可以用事件总线，借助guava中的EventBus来实现是吗

> 作者回复：是的。这种主要是处理微服务内部不同聚合之间的领域事件。主要是因为在处理新增和修改数据的时候，一个操作是对一个聚合整理的数据进行操作的。这样可以避免聚合之间的数据不一致的情况。

2020-1-16

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

> 作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚

合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

作者回复：JPA可以自动创建的。

2020-1-13

uangguan

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️🅰️

作者回复：过来人🅰️🅰️。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码

中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么？

作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰🅰

作者回复：🅰🅰，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图

一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

> 作者回复：是的，DDD不是一个人的事情。

2020-1-2

观弈道人

终于等到了，谢谢大佬。

2020-1-2

和光同尘

请问数据对账机制，是如何知道有数据缺损，不一致的？再者，发现不一致的数据如何纠正。3Q

> 作者回复：可以基于源端持久化和目的端持久化数据来比对，发现异常数据。处理方式有很多，发现问题后可以重传或者人工处理。

2020-1-2

Jxin

1.感谢栏主的demo。劳烦了。

2.毕竟是用于大概了解ddd的demo，各中实现比较简单，并没有很讲究设计原则。如果有下一个专栏再探讨。

3.认真看完栏主代码实现，跟自己的实现规范差别还是挺多的。各有利弊就不多赘述了。

4.期待下次相见。

> 作者回复：DDD的主要目标是为了边界划分和解耦，而实现这个目标有很多的手段。目标只有一个，而手段会有很多个，只要能达到这个目标，可以选择适合自己的手段，咱们的主要区别可能在手段上有差异。有机会向你学习哈。

2020-1-2

深山小书童

终于等到了，欧老师威武

2020-1-2

二康

谢谢老师

2020-1-2

切糕

老师，您好！我们这边有个比较怪异的场景，想请教下老师：
背景介绍：我们主要是做分期账单的业务，整体分为两部分，一部分是订单、一部分是账单。订单中包含渠道、收款人、付款人信息，而这些信息因渠道侧不同字段会有很大差异。后面的账单基本是根据订单生成的，初始账单的“渠道、收款人、付款人”信息是一致的，但后续账单可能会变更收款人信息。
目前状况：我们把订单、账单分成两个聚合，这个时候“渠道、收款人、付款人”通过抽象ID维护在订单聚合，但是账单这边也需要“渠道、收款人、付款人”实体，这个时候复用好像又不太对，因为这个实体在订单聚合内，账单聚合操作依赖订单聚合中的实体，感觉很怪。
个人见解：
1.订单的生命周期主要是为了生成账单，一但账单生成订单的生命周期结束，所以订单中的“渠道、收款人、付款人”以值对象JSON存储下来，而不同渠道的字段不一样，在和不同渠道业务交互的规则中去处理。
2.账单聚合中维护“渠道、收款人、付款人”实体对象，因存在变更收款人操作等操作，所以收款人比较适合做实体，而渠道和付款人信息应该是和收款人属于同一级别，所以都认为是实体。

对这一块还是比较疑惑，希望老师能给些建议。

> 作者回复：我建议你再设计两个聚合：渠道和人员（暂且叫人员吧，你可以根据你的业务场景来命名）聚合。人员聚合主要提供人员的新增和修改，而这个人员数据可以被订单或账单聚合以值对象的方式来引用，在值对象引用时你可以根据业务类型将人员设置为收款人或者付款人。这样人员信息可以在人员聚合集中维护，它可以同时被多个其它的聚合以值对象的形式整体引用。

2020-1-15

Geek_74f563

老师可以把数据库表结构的脚本上传上去吗？ 哈哈

> 作者回复：JPA可以自动创建的。

2020-1-13

uangguan

uangguan

老师，LeaveRepositoryInterface接口的save方法，为什么传入的是LeavePO而不是Leave？在LeaveDomainService层里直接调用LeaveRepositoryInterface.save(Leave leave)保存领域实体不是更合理吗？这样在LeaveDomainService层就不需要知道LeavePO的存在了。

> 作者回复：你这个想法挺好的。基础层的对象是可以为所有层提供服务的，把PO拿出来主要还是希望将DO和PO的转换统一放在工厂服务中。如果用leave的DO对象做完save的参数，在save仓储实现内，一样还是需要转换的。

2020-1-13

宋承展

老师好，请教一下在查询列表这种交易中，一般我们会返回翻页通信区给前端，这个翻页通信区，一般放在哪一层的。。。我看在领域服务层里，都是只是返回了实体的list，对于这种翻页通信区，没说明，能否指导一下？谢谢

作者回复：复杂的查询可以不走领域层，你可以定义一个查询应用服务，按照传统三层架构分页查询方式就可以了。

2020-1-11

SpEcIAL_ID

赞赞赞，很多共鸣，🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋🙋

作者回复：过来人🙋。

2020-1-10

Geek_aa8017

老师，你之前的章节说接口层到应用层参数的转换是dto转do的，但现在看代码怎么是dto直接转实体传给应用层的

作者回复：实体是DO的一种呀。

2020-1-9

高忠宝

老师：聚合leave中Apprever中的fromPerson（Person person)这个方法是不是和person聚会耦合了？把这个转换放在应用层是否是更合适?

作者回复：这个确实有一定的聚合耦合度。但是考虑到如果person和leave聚合拆分，我们还会增加一个person的DTO对象来接受从person返回的对象。如果两者结构能保持一致，拆分后fromPerson也是不需要做出代码调整的。所以为了不增加代码的复杂度，没有考虑这种返回类型的对象解耦。这种转换放应用层也是可以的。

2020-1-7

snailiu

感觉`PersonDomainService`中的`findNextApprover`和`findFirstApprover`有点不妥，Approver是Leave聚合中的概念，而不是Person聚合中的概念，从两个函数的具体实现来看，也和Approver没有关系，而只是根据personId和maxLeaderLevel获取Person对象。

作者回复：这两个服务本质上是根据申请人找出人员的上级关系，它就是人员组织关系的业务职能的呀。

2020-1-7

观弈道人

欧老师， 源码里是在domainservice依赖注入仓储，而文章里是通过实体注入仓储对象，不一致～

作者回复：可否具体说一下是那个地方?

2020-1-6

zj

我看了github上的代码，有点问题，对于LeavePO和List《ApprovalInfoPO》这样的一对多关联持久化问题。代码中没有看到approvqlInfoPO.setLeaveId（），因为必须先保存LeavePO才会有leaveId。因此持久化Leave操作不能简单使用LeaveRepositoryImpl.save方法

作者回复：感谢提醒，已补充。

```
public void save(LeavePO leavePO) {
//persist leave entity
leaveDao.save(leavePO);
//set leave_id for approvalInfoPO after save leavePO
leavePO.getHistoryApprovalInfoPOList().stream().forEach(approvalInfoPO ->
approvalInfoPO.setLeaveId(leavePO.getId()));
approvalInfoDao.saveAll(leavePO.getHistoryApprovalInfoPOList());
}
```

2020-1-6

hk

老师你好，问个小白问题：创表SQL需要我们自己生成出来是么?

作者回复：看你用什么样的持久化组件。有的可以根据PO自动创建。

2020-1-3

zj

老师为什么我看代码里面Leave这个实体类没采用直接依赖仓储层的方式呢

> 作者回复：因为聚合数据要作为整体来修改和创建，持久化的工作统一放到工厂服务里去实现了。

2020-1-3

sqyao

期待已久的ddd demo项目终于出炉了，感谢欧老师！🅰🅰

> 作者回复：🅰🅰，久等了。

2020-1-3

南山

感谢老师的长篇分享，对其中一句深表赞同，不自觉的调用其他聚合的东西，不自觉有两种，一种是老师说的图一时快，另一种比较难受，就是不知道ddd或者固执己见，达不成一致，统一思想开发风格很重要！

> 作者回复：是的，DDD不是一个人的事情。

2020-1-2

观弈道人

终于等到了，谢谢大佬。

2020-1-2

和光同尘

请问数据对账机制，是如何知道有数据缺损，不一致的？再者，发现不一致的数据如何纠正。3Q

> 作者回复：可以基于源端持久化和目的端持久化数据来比对，发现异常数据。处理方式有很多，发现问题后可以重传或者人工处理。

2020-1-2

Jxin

1.感谢栏主的demo。劳烦了。

2.毕竟是用于大概了解ddd的demo，各中实现比较简单，并没有很讲究设计原则。如果有下一个专栏再探讨。

3.认真看完栏主代码实现，跟自己的实现规范差别还是挺多的。各有利弊就不多赘述了。

4.期待下次相见。

> 作者回复：DDD的主要目标是为了边界划分和解耦，而实现这个目标有很多的手段。目标只有一个，而手段会有很多个，只要能达到这个目标，可以选择适合自己的手段，咱们的主要区别可能在手段上有差异。有机会向你学习哈。

2020-1-2

深山小书童

终于等到了，欧老师威武

2020-1-2

二康

谢谢老师

2020-1-2

阿神

代码简洁易懂，很赞

> 作者回复：谢谢🅰🅰

2020-1-2